Scientific & Technical Information Center
USPTO

# STIC Search Report
## EIC 2100

## STIC Database Tracking Number: 131531

| | |
|---|---|
| TO: James Kerveros<br>Location: 4D21<br>Art Unit : 2133<br>Friday, September 03, 2004<br><br>Case Serial Number: 09/647296 | From: David Holloway<br>Location: EIC 2100<br>PK2-4B30<br>Phone: 308-7794<br><br>david.holloway@uspto.gov |

## Search Notes

Dear Examiner Kerveros,

Attached please find your search results for above-referenced case.
Please contact me if you have any questions or would like a re-focused search.

David

ALBERT!
FOR YOUR INFO
SEARCH: FOCUS
IN SEARCHPLUS

BEST AVAILABLE COPY

Scientific & Technical Information Center
STIC
USPTO

# STIC EIC 2100
# Search Request Form

131531
1

Today's Date:

**What date would you like to use to limit the search?**

Priority Date: Other:

Name JAMES KERVEROS

AU 2133 Examiner # 77270

Room # PK2 4D21 Phone 305-1081

Serial # 09/647,296

**Format for Search Results (Circle One):**

PAPER DISK [✓ EMAIL]

**Where have you searched so far?**

[✓ USP] [✓ DWPI] [✓ EPO] [✓ JPO] ACM IBM TDB

IEEE INSPEC SPI Other ______

**Is this a "Fast & Focused" Search Request? (Circle One) YES [✓ NO]**
A "Fast & Focused" Search is completed in 2-3 hours (maximum). The search must be on a very specific topic and meet certain criteria. The criteria are posted in EIC2100 and on the EIC2100 NPL Web Page at http://ptoweb/patents/stic/stic-tc2100.htm.

What is the topic, novelty, motivation, utility, or other specific details defining the desired focus of this search? Please include the concepts, synonyms, keywords, acronyms, definitions, strategies, and anything else that helps to describe the topic. Please attach a copy of the abstract, background, brief summary, pertinent claims and any citations of relevant art you have found.

SEE ATTACHED CLAIMS
UNDERLINE PHRASES.

**Best Available Copy**

STIC Searcher David Holloway Phone 308-7794

Date picked up 9-7-04 Date Completed 9-7-04

sira
Search and Information Resources Administration

DIALOG

HP Tech Bulletin 6 Yrs 477 61

10/647,926 JAMES KERVEROS PK2/4D21

In the Claims:

1. A method for testing a processor using random code generation, the method, comprising:

defining an initial state of the processor, the initial state comprising data locations in the processor, wherein defining the initial state comprises, for each data location:

creating an uncommitted value,

setting an initial state pointer to the uncommitted value, and

setting a current state pointer to the uncommitted value;

generating a random instruction for each data location;

executing the random instructions, the executed random instructions comprising a random program, wherein executing the random instructions produces current values for the data locations, and wherein the current values can be one of committed, uncommitted, and deferred;

determining a length of the random program;

if the random program length equals a desired length, executing a commit V (value) routine; and

if the random program length is less than the desired length, repeating the generating, executing, and determining steps.

2. The method of claim 1, wherein executing the commit V (value) routine, comprises:

determining a state of a value V for each data location;

if the state of the value V is committed, ending the routine as to that value;

if the state of the value V is deferred:

determining an instruction that produced the value V, and

calling an execute instruction routine; and

if the state of the value V is uncommitted:

generating a random number X, and

calling a commit V, X routine.

3. The method of claim 2, wherein calling an execute instruction routine, comprises:

for each input value to an instruction, executing the commit V (value) routine;

performing an instruction set architecture (ISA) operation for the instruction, wherein an ISA result value is generated;

setting an output value of the instruction equal to the ISA result; and

calling a demote instruction routine.

9/4/04

Ex A

# TESTING A PROCESSOR USING A RANDOM CODE GENERATOR

**Cross Reference To Related Application(s)**

This application is a continuation application of copending application number 09/510,371, filed February 22, 2000, which is hereby incorporated by reference in its entirety.

INSERT (See Ex A)

**Technical Field**

The technical field is design verification of central processing units that execute programs according to Instruction Set Architectures.

**Background**

Instruction set architectures (ISA) specify how central processing units (CPU) execute programs. The programs include instructions stored in memory. Instructions are typically simple, primitive operations such as add, branch on condition, load from memory, and store to memory.

To provide for software compatibility with legacy computer systems, modern CPUs should adhere to the ISA with minimal defects. To achieve this goal, computer designers could verify the correct behavior of each kind of instruction in isolation. However, this may not be sufficient because, in an effort to improve performance, modern CPUs may overlap the execution of numerous instructions. In this environment, some defects may be exposed only when specific combinations of instructions are executed in sequence. Thus, the designer may desire to test every combination or sequence of instructions. However, modern CPUs typically execute over 100 instructions, each of which can include numerous options. Mechanically enumerating all of these possible instructions sequences may not feasible. Even if all possible instruction sequences were enumerated, the designer's test regime may be insufficient because many defects require more than just a specific sequence of instructions. In particular, additional events, such as operand data patterns, memory address collisions between instructions, the state of CPU structures such as cache memories, and stimulus received by the CPU can pre condition defects.

One of the most important tools used by a CPU designer to address these challenges is a Random Code Generator (RCG). The RCG creates a random software program that is used during the CPU design and prototype verification process to generate billions of random instructions to test the CPU.

```
Set     Items   Description
S1    5965888   TEST OR TESTING OR TESTS OR TESTED OR VERIF? OR DEBUG? OR -
             EMULAT?
S2      18709   RCG OR RPG OR (PSEUDORANDOM OR RANDOM)()(PROGRAM? OR CODE?)
S3   12116169   COMMIT? OR UNCOMMIT? OR DEFER? OR VALUE?
S4      16575   (LENGTH? OR SIZE? OR BYTE?)(N)(RANDOM()PROGRAM? OR RANDOM(-
             )CODE? OR CODE? OR PROGRAM?) OR CODELENGTH?
S5        403   S1 (9N) S2
S6         20   S5 (S) (S3 OR S4)
S7    2580710   CPU OR CPUS OR VLSI OR ASIC OR CHIP OR CHIPS OR MICROCHIP -
             OR PRESILICON OR POSTSILICON OR PROCESSOR?
S8        106   S5 AND S7
S9         20   S5(10N)S7
S10        38   S9 OR S6
S11        28   RD (unique items)
S12        25   S11 NOT PY>2000
S13        25   S12 NOT PD=20000222:20030222
S14        25   S13 NOT PD=20030222:20040922
File 275:Gale Group Computer DB(TM) 1983-2004/Sep 03
         (c) 2004 The Gale Group
File  47:Gale Group Magazine DB(TM) 1959-2004/Sep 03
         (c) 2004 The Gale group
File  75:TGG Management Contents(R) 86-2004/Aug W4
         (c) 2004 The Gale Group
File 636:Gale Group Newsletter DB(TM) 1987-2004/Sep 03
         (c) 2004 The Gale Group
File  16:Gale Group PROMT(R) 1990-2004/Sep 03
         (c) 2004 The Gale Group
File 624:McGraw-Hill Publications 1985-2004/Sep 02
         (c) 2004 McGraw-Hill Co. Inc
File 484:Periodical Abs Plustext 1986-2004/Aug W4
         (c) 2004 ProQuest
File 613:PR Newswire 1999-2004/Sep 03
         (c) 2004 PR Newswire Association Inc
File 813:PR Newswire 1987-1999/Apr 30
         (c) 1999 PR Newswire Association Inc
File 141:Readers Guide 1983-2004/Jul
         (c) 2004 The HW Wilson Co
File 239:Mathsci 1940-2004/Oct
         (c) 2004 American Mathematical Society
File 370:Science 1996-1999/Jul W3
         (c) 1999 AAAS
File 696:DIALOG Telecom. Newsletters 1995-2004/Sep 02
         (c) 2004 The Dialog Corp.
File 553:Wilson Bus. Abs. FullText 1982-2004/Jul
         (c) 2004 The HW Wilson Co
File 621:Gale Group New Prod.Annou.(R) 1985-2004/Sep 03
         (c) 2004 The Gale Group
File 674:Computer News Fulltext 1989-2004/Aug W3
         (c) 2004 IDG Communications
File  88:Gale Group Business A.R.T.S. 1976-2004/Sep 02
         (c) 2004 The Gale Group
File 369:New Scientist 1994-2004/Aug W4
         (c) 2004 Reed Business Information Ltd.
File 160:Gale Group PROMT(R) 1972-1989
         (c) 1999 The Gale Group
File 635:Business Dateline(R) 1985-2004/Sep 03
         (c) 2004 ProQuest Info&Learning
File  15:ABI/Inform(R) 1971-2004/Sep 03
         (c) 2004 ProQuest Info&Learning
File   9:Business & Industry(R) Jul/1994-2004/Sep 02
         (c) 2004  The Gale Group
File  13:BAMP 2004/Aug W4
         (c) 2004  The Gale Group
File 810:Business Wire 1986-1999/Feb 28
         (c) 1999 Business Wire
File 610:Business Wire 1999-2004/Sep 03
         (c) 2004 Business Wire.
```

```
File 647:CMP  Computer Fulltext 1988-2004/Aug W4
         (c) 2004 CMP Media, LLC
File  98:General Sci Abs/Full-Text 1984-2004/Jul
         (c) 2004 The HW Wilson Co.
File 148:Gale Group Trade & Industry DB 1976-2004/Sep 03
         (c)2004 The Gale Group
File 634:San Jose Mercury  Jun 1985-2004/Sep 02
         (c) 2004 San Jose Mercury News
```

**14/3,K/1 (Item 1 from file: 275)**
DIALOG(R)File 275:Gale Group Computer DB(TM)

02104514 SUPPLIER NUMBER: 19758438 (USE FORMAT 7 OR 9 FOR FULL TEXT)
**Electrical verification of the HP PA 8000 processor. (Product Development)(Technical)**
Ljung, David J.; Bockhaus, John W.; Bhatia, Rohit; Ramsey, C. Michael;
Butler, Joseph R.
Hewlett-Packard Journal, v48, n4, p32(8)
August, 1997
DOCUMENT TYPE: Technical ISSN: 0018-1153 LANGUAGE: English
RECORD TYPE: Fulltext; Abstract
WORD COUNT: 7104 LINE COUNT: 00539

... volume of random testing can be accomplished.

The electrical verification of the HP PA 8000 **CPU** relied upon the following sources for test cases:

* Directed handwritten **tests**
* Focused random **tests** targeted at specific **CPU** functions
* **Random code** generators
* Library of worst-case **tests** for previous bugs
* HP-UX* application code.

In most instances, these test cases were checking...self-explanatory.

The second type of failure is a final-state error, generally produced by **random code** generators. **Random code** generators produce **tests** that consist of initial **CPU** state, a sequence of assembly instructions, and an expected final state. When a test terminates...

**14/3,K/2 (Item 2 from file: 275)**
DIALOG(R)File 275:Gale Group Computer DB(TM)

02104513 SUPPLIER NUMBER: 19758437 (USE FORMAT 7 OR 9 FOR FULL TEXT)
**Functional verification of the HP PA 8000 processor. (Product Information)(Technical)**
Mangelsdorf, Steven T.; Gratias, Raymond P.; Blumberg, Richard M.; Bhatia, Rohit
Hewlett-Packard Journal, v48, n4, p22(10)
August, 1997
DOCUMENT TYPE: Technical ISSN: 0018-1153 LANGUAGE: English
RECORD TYPE: Fulltext; Abstract
WORD COUNT: 8154 LINE COUNT: 00655

... these machines under the control of HP Task Broker.(1) Each job ran several thousand **test** cases that were generated using a specific **pseudorandom code** generator control file.

Multiprocessor **Testing**

Multiprocessor **testing** was a key focus area. We wrote emulators for additional **processors** and the I/O adapter, which share the memory bus. It was only necessary to...

...the emulators could initiate transactions that were likely to cause interactions.

Coverage Improvement

Improving the **test** coverage of our **pseudorandom code** generator was an ongoing activity. The pseudorandom code generator has hundreds of adjustable **values** , or knobs, in its control file, which can be varied to focus the generated test...

...monitor the quality of the files generated.

We did this in two ways. First, our **pseudorandom code** generator itself reported statistics on the **test** cases generated with a given control file. A good example is the frequency of traps...

...queues, so having too many traps in a case effectively shortens it and reduces its **value** . We made use of instrumentation like this to steer the generation of control files.

Feedback...

**14/3,K/8 (Item 8 from file: 275)**
DIALOG(R)File 275:Gale Group Computer DB(TM)

01541313 SUPPLIER NUMBER: 12653766 (USE FORMAT 7 OR 9 FOR FULL TEXT)
**RPG development. (Harland Software's RPG Development System) (Brief Article) (Product Announcement)**
MIDRANGE Systems, v5, n19, p53(1)
Oct 13, 1992
DOCUMENT TYPE: Product Announcement ISSN: 1041-8237 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT
WORD COUNT: 202 LINE COUNT: 00016

... files. All RPG II and RPG III operation codes, including string operations, are implemented except **commitment** control and multiple device support. The CALL/PARM feature can call to any depth up...

...can be executed and external subroutines written in other languages can be linked with the **RPG** program.

Capability for source-level **debugging** is provided with the Harland **RPG** compiler. Standard PC **debuggers** , such as Microsoft Codeview or Symdeb and Borland Turbo Debugger, can be used to debug...

```
Set     Items   Description
S1    6088455   TEST OR TESTING OR TESTS OR TESTED OR VERIF? OR DEBUG? OR -
             EMULAT?
S2       2693   RCG OR RPG OR (PSEUDORANDOM OR RANDOM)()(PROGRAM? OR CODE?)
S3    3929464   COMMIT? OR UNCOMMIT? OR DEFER? OR VALUE?
S4      12515   (LENGTH? OR SIZE? OR BYTE?)(N)(RANDOM()PROGRAM? OR RANDOM(-
             )CODE? OR CODE? OR PROGRAM?) OR CODELENGTH?
S5        341   S1 AND S2
S6         57   S5 AND (S3 OR S4)
S7     814890   CPU OR CPUS OR VLSI OR ASIC OR CHIP OR CHIPS OR MICROCHIP -
             OR PRESILICON OR POSTSILICON OR PROCESSOR?
S8          0   S6 AND S7
S9         39   RD S6 (unique items)
S10        25   S9 NOT PY>2000
S11        25   S10 NOT PD>20000222
File   8:Ei Compendex(R) 1970-2004/Aug W4
         (c) 2004 Elsevier Eng.  Info. Inc.
File  35:Dissertation Abs Online 1861-2004/Jul
         (c) 2004 ProQuest Info&Learning
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jul 12
         (c) 2004 EBSCO Publishing
File  65:Inside Conferences 1993-2004/Aug W5
         (c) 2004 BLDSC all rts. reserv.
File   2:INSPEC 1969-2004/Aug W4
         (c) 2004 Institution of Electrical Engineers
File  94:JICST-EPlus 1985-2004/Aug W1
         (c)2004 Japan Science and Tech Corp(JST)
File 111:TGG Natl.Newspaper Index(SM) 1979-2004/Sep 03
         (c) 2004 The Gale Group
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File   6:NTIS 1964-2004/Aug W4
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File 144:Pascal 1973-2004/Aug W4
         (c) 2004 INIST/CNRS
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Aug W5
         (c) 2004 Inst for Sci Info
File  62:SPIN(R) 1975-2004/Jul W1
         (c) 2004 American Institute of Physics
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Jul
         (c) 2004 The HW Wilson Co.
File  95:TEME-Technology & Management 1989-2004/Jun W1
         (c) 2004 FIZ TECHNIK
```

**11/5/2 (Item 2 from file: 8)**

DIALOG(R)File 8:Ei Compendex(R)

04479635 E.I. No: EIP96083290138

**Title:** **Converter synthesis of** pseudorandom codes **for self-** testing **of digital devices satisfying IEEE standards**

Author: Bykov, Yu.V.; Yarmolik, V.N.

Corporate Source: Belorusskij Gosudarstvennyj Univ Informatiki i Radioelektroniki, Minsk, Belorus

Source: Avtomatika i Telemekhanika n 4 Apr 1996. p 148-154

Publication Year: 1996

CODEN: AVTEAI

Language: Russian

Document Type: JA; (Journal Article) Treatment: A; (Applications); T; (Theoretical)

Journal Announcement: 9610W4

Abstract: Methods are suggested for synthesis of digital circuits used for the formation of **testing** actions oriented to the application in self-**testing** digital circuits containing the standard diagnostic equipment. Peculiarities of the synthesized devices are analyzed, and the devices are compared with the known converters. The suggested methods allow to design devices with the following positive properties: (1) memory elements of the BS-cells are used in the design of **pseudorandom codes** converters what allows to reduce the number of assumed sequences of independent bits and (2) synthesized circuits have the constant delay time **value** with which the next symbol of the **test** sequence is formed. 17 Refs.

Descriptors: Electric network synthesis; Codes (symbols); Probability; Reliability; **Testing** ; Standards; Digital circuits

Identifiers: Converters; **Pseudorandom codes**

Classification Codes:

703.1.2 (Electric Network Synthesis)

703.1 (Electric Networks); 723.2 (Data Processing); 922.1 (Probability Theory); 902.2 (Codes & Standards); 721.3 (Computer Circuits)

703 (Electric Circuits); 723 (Computer Software); 922 (Statistical Methods); 902 (Engineering Graphics & Standards); 721 (Computer Circuits & Logic Elements)

70 (ELECTRICAL ENGINEERING); 72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS); 90 (GENERAL ENGINEERING)

```
Set     Items   Description
S1     547152   TEST OR TESTING OR TESTS OR TESTED OR VERIF? OR DEBUG? OR -
             EMULAT?
S2       1551   RCG OR RPG OR RANDOM()(PROGRAM? OR CODE?)
S3     664141   COMMIT? OR UNCOMMIT? OR DEFER? OR VALUE?
S4       7793    (LENGTH? OR SIZE? OR BYTE?)(N)(RANDOM()PROGRAM? OR RANDOM(-
             )CODE? OR CODE? OR PROGRAM?) OR CODELENGTH?
S5         42   S1 (12N) S2
S6          2   S5 (10N) (S3 OR S4)
S7          0   S5 AND IC=H02H?
S8          3   S5 AND IC=(G06F-009? OR G06F-011?)
S9         10   S5 AND IC=G06F?
S10        12   S9 OR S6
S11        12   IDPAT (sorted in duplicate/non-duplicate order)
S12        12   IDPAT (primary/non-duplicate records only)
S13       371   PSEUDORANDOM()(PROGRAM OR PROGRAMS OR CODE OR CODES)
S14         0   S13(12N)S1
S15        30   S5 AND (REPEAT? OR ITERAT? OR REITERAT? OR RETEST?)
S16        22   S15 NOT S10
S17         6   S16 (12N) (VLSI OR CPU OR CHIP OR MICROCHIP? OR PROCESSOR?
             OR ASIC? OR CPUS OR CHIPS)
File 348:EUROPEAN PATENTS 1978-2004/Aug W05
         (c) 2004 European Patent Office
File 349:PCT FULLTEXT 1979-2002/UB=20040826,UT=20040819
         (c) 2004 WIPO/Univentio
```

```
Set     Items   Description
S1     547152   TEST OR TESTING OR TESTS OR TESTED OR VERIF? OR DEBUG? OR -
             EMULAT?
S2       1551   RCG OR RPG OR RANDOM()(PROGRAM? OR CODE?)
S3     664141   COMMIT? OR UNCOMMIT? OR DEFER? OR VALUE?
S4       7793   (LENGTH? OR SIZE? OR BYTE?)(N)(RANDOM()PROGRAM? OR RANDOM(-
             )CODE? OR CODE? OR PROGRAM?) OR CODELENGTH?
S5         42   S1 (12N) S2
S6          2   S5 (10N) (S3 OR S4)
S7          0   S5 AND IC=H02H?
S8          3   S5 AND IC=(G06F-009? OR G06F-011?)
S9         10   S5 AND IC=G06F?
S10        12   S9 OR S6
S11        12   IDPAT (sorted in duplicate/non-duplicate order)
S12        12   IDPAT (primary/non-duplicate records only)
File 348:EUROPEAN PATENTS 1978-2004/Aug W05
         (c) 2004 European Patent Office
File 349:PCT FULLTEXT 1979-2002/UB=20040826,UT=20040819
         (c) 2004 WIPO/Univentio
```

**12/3,K/11     (Item 11 from file: 349)**
DIALOG(R)File 349:PCT FULLTEXT

00284670
**DIRECT-SEQUENCE SPREAD-SPECTRUM ULTRASONIC TESTING DEVICE**
**DISPOSITIF D'ESSAI ULTRASONORE A ETALEMENT DU SPECTRE EN SEQUENCE DIRECTE**
Patent Applicant/Assignee:
  IOWA STATE UNIVERSITY RESEARCH FOUNDATION INC,
Inventor(s):
  PAPADAKIS Emmanuel P,
  RUSSELL Steve F,
  WORMLEY Samuel J,
Patent and Priority Information (Country, Number, Date):
  Patent:               WO 9502819 A1 19950126
  Application:          WO 94US7857 19940713  (PCT/WO US9407857)
  Priority Application: US 9391179 19930713
Designated States:
(Protection type is "patent" unless otherwise stated - for applications prior to 2004)
  CA AT BE CH DE DK ES FR GB GR IE IT LU MC NL PT SE
Publication Language: English
Fulltext Word Count: 6895

Fulltext Availability:
  Claims

Claim
...  36 wherein the signature signal is compared with a signature signal previously obtained for the **test** object.

40 The method of claim 35 wherein the pseudo- **random** **code** has a **code** **length** which is sufficiently high to produce spectral components in the electrical input signal which are...

```
Set     Items   Description
S1     723147   TEST OR TESTING OR TESTS OR TESTED OR VERIF? OR DEBUG? OR -
             EMULAT?
S2        995   RCG OR RPG OR RANDOM()(PROGRAM? OR CODE?)
S3    1361481   COMMIT? OR UNCOMMIT? OR DEFER? OR VALUE?
S4       5888   (LENGTH? OR SIZE? OR BYTE?)(N)(RANDOM()PROGRAM? OR RANDOM(-
             )CODE? OR CODE? OR PROGRAM?) OR CODELENGTH?
S5        125   S1 AND S2
S6         21   S5 AND (S3 OR S4)
S7          2   S5 AND IC=H02H?
S8         22   S6 OR S7
S9         22   IDPAT (sorted in duplicate/non-duplicate order)
S10        22   IDPAT (primary/non-duplicate records only)
S11     11026   MC=(S01-G01 OR S01-G01A5 OR T01-G02A2D OR T01-G07A)
S12         9   S11 AND S2
S13         5   S12 NOT S8
File 347:JAPIO Nov 1976-2004/Apr(Updated 040802)
         (c) 2004 JPO & JAPIO
File 350:Derwent WPIX 1963-2004/UD,UM &UP=200456
         (c) 2004  Thomson Derwent
```

10/5/21 **(Item 21 from file: 347)**
DIALOG(R)File 347:JAPIO

06260446 **Image available**
METHOD FOR FAILURE ANALYSIS

| | |
|---|---|
| PUB. NO.: | 11-202026 [JP 11202026 A] |
| PUBLISHED: | July 30, 1999 (19990730) |
| INVENTOR(s): | KAMATA SATOSHI<br>IKETANI TOYOHITO<br>KOMINAMI ATSUSHI |
| APPLICANT(s): | HITACHI LTD<br>HITACHI INFORMATION TECHNOLOGY CO LTD |
| APPL. NO.: | 10-004489 [JP 984489] |
| FILED: | January 13, 1998 (19980113) |
| INTL CLASS: | G01R-031/28 |

## ABSTRACT

PROBLEM TO BE SOLVED: To locate a failure in an LSI to be **tested** using BEST(built-in self **test** ) method by subdividing all **test** patterns equally, reading out an SA (encoded compressor) **value** upon finishing application of pattern of each Gr and comparing it with an expected code.

SOLUTION: At first, a system clock SC is driven to prepare for generation of pseudorandom numbers from an **RPG** (random number generator) 20 in an LSI 9 to be **tested** . When scan-in clock is driven continuously by total number of patterns PA, the **RPG** 20 generates pseudorandom numbers sequentially and input latches (a) are shifted sequentially before the pseudorandom numbers are applied to a circuit 19 to be **tested** . Output data from the LSI 9 is captured by each output latch (b) and a scan-out clock SOCK is driven to scan out an SA 21. When the SC is driven upon finishing clock control of total number of patterns PA, a comparator (d) compares an output **value** with an expected code and the comparison results are described in a fail memory 12.

```
 10/5/5     (Item 5 from file: 350)
DIALOG(R)File 350:Derwent WPIX
(c) 2004  Thomson Derwent. All rts. reserv.

014113747     **Image available**
WPI Acc No: 2001-597959/200168
XRPX Acc No: N01-445848
   Random   code  generation for functional operation  testing  in
  integrated circuit design, involves generating random weightings for
  predefined control  values  assigned to instruction type to be included
  in  random   code
Patent Assignee: HEWLETT-PACKARD CO (HEWP  ); HEWLETT-PACKARD DEV CO LP
  (HEWP  )
Inventor: BRUMMEL K P
Number of Countries: 002  Number of Patents: 003
Patent Family:
Patent No      Kind   Date    Applicat No    Kind   Date     Week
DE 10056825    A1  20010705  DE 1056825       A   20001116  200168  B
DE 10056825    C2  20031009  DE 1056825       A   20001116  200366
US 6678853     B1  20040113  US 99466503      A   19991217  200405

Priority Applications (No Type Date): US 99466503 A 19991217
Patent Details:
Patent No  Kind Lan Pg   Main IPC    Filing Notes
DE 10056825   A1      8 G06F-007/58
DE 10056825   C2        G06F-007/58
US 6678853    B1        G01R-031/28

Abstract (Basic): DE 10056825 A1
        NOVELTY - Random weightings are generated for preset control
    values  assigned to instruction type to be included in the  random
    code  to be generated. The  random   code  with instruction type having
    its statistical probability corresponding to the generated random
    weightings, is generated.
        DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the
    following:
        (a)  Random   code  generator;
        (b) Recording medium storing  random   code  generation program
        USE - For generating  random   code  for functional operation
    testing  of integrated circuit design e.g. for microprocessor.
        ADVANTAGE - Facilitates to generate  random   code  set using
    single control type and facilitates to control code generator
    effectively and exactly using simple technique.
        DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of
    random   code  generator. (Drawing includes non-English language text).

        pp; 8 DwgNo 2/4
Title Terms: RANDOM; CODE; GENERATE; FUNCTION; OPERATE;  TEST ; INTEGRATE;
  CIRCUIT; DESIGN; GENERATE; RANDOM; WEIGHT; PREDEFINED; CONTROL;  VALUE ;
  ASSIGN; INSTRUCTION; TYPE; RANDOM; CODE
Derwent Class: T01; T05
International Patent Class (Main): G01R-031/28; G06F-007/58
International Patent Class (Additional): G06F-011/263; G07C-015/00
File Segment: EPI
```

```
10/5/10     (Item 10 from file: 350)
DIALOG(R)File 350:Derwent WPIX
(c) 2004  Thomson Derwent. All rts. reserv.

012010495     **Image available**
WPI Acc No: 1998-427405/199836
XRPX Acc No: N98-333631
  Processor operation  testing  method using  random   code  generator for
  CPU, microprocessor and VLSI circuit - involves comparing actual
  operation state of processor with standard  value , based on which
  functional mode of processor is judged
Patent Assignee: HEWLETT-PACKARD CO (HEWP  )
Inventor: BROCKMANN R C; BRUMMEL K
Number of Countries: 001  Number of Patents: 001
Patent Family:
Patent No     Kind   Date    Applicat No    Kind   Date     Week
US 5784550     A   19980721  US 96739244     A   19961029  199836  B

Priority Applications (No Type Date): US 96739244 A 19961029
Patent Details:
Patent No  Kind Lan Pg   Main IPC    Filing Notes
US 5784550    A     11 G06F-011/00

Abstract (Basic): US 5784550 A
        The method involves generating the  test  case from a  random
    code  generator corresponding to the interrupt signal generation. The
    test  case is simulated by a simulator to determine the standard
    operation state of a processor.
        Then, the  test  case is executed by a processor, from which the
    actual operation state of the processor is detected. The actual
    operation state is compared with the standard  value , based on which
    the functional mode is judged.
        ADVANTAGE - Enables failure detection in processor, quickly.
    Enables execution of multiple traps with same instruction. Reduces
    number of instructions involved in functional  testing . Avoids
    reduction in  test  case length. Handles multiple interrupt conditions
    easily with priority order.
        Dwg.4/5
Title Terms: PROCESSOR; OPERATE;  TEST ; METHOD; RANDOM; CODE; GENERATOR;
  CPU; MICROPROCESSOR; VLSI; CIRCUIT; COMPARE; ACTUAL; OPERATE; STATE;
  PROCESSOR; STANDARD;  VALUE ; BASED; FUNCTION; MODE; PROCESSOR; JUDGEMENT
Derwent Class: T01
International Patent Class (Main): G06F-011/00
File Segment: EPI
```

```
Set      Items   Description
S1          13   AU=(MANGELSDORF S? OR MANGELSDORF, S?)
S2           1   S1 AND IC=H02H?
File 347:JAPIO Nov 1976-2004/Apr(Updated 040802)
         (c) 2004 JPO & JAPIO
File 350:Derwent WPIX 1963-2004/UD,UM &UP=200456
         (c) 2004  Thomson Derwent
```

```
 2/9/1     (Item 1 from file: 350)
DIALOG(R)File 350:Derwent WPIX
(c) 2004  Thomson Derwent. All rts. reserv.

014143037     **Image available**
WPI Acc No: 2001-627248/200173
XRPX Acc No: N01-467631
  Management of non-fixed register values during random program generation
  e.g. for design verification of CPUs, requires marking a value state as
  non-fixed when the content is unknown and when any desired content can be
  generated
Patent Assignee: HEWLETT-PACKARD CO (HEWP  ); MANGELSDORF S T (MANG-I);
  HEWLETT-PACKARD DEV CO LP (HEWP  )
Inventor:  MANGELSDORF S T
Number of Countries: 002  Number of Patents: 003
Patent Family:
Patent No      Kind   Date    Applicat No     Kind   Date     Week
DE 10058371     A1  20010906  DE 10058371      A   20001124  200173  B
US 6671664      B1  20031230  US 2000510371    A   20000222  200402
US 20040153805  A1   20040805  US 2000510371    A   20000222  200452
                              US 2003647296    A   20030826

Priority Applications (No Type Date): US 2000510371 A 20000222; US
  2003647296 A 20030826
Patent Details:
Patent No  Kind Lan Pg   Main IPC    Filing Notes
DE 10058371   A1    23 G06F-011/263
US 6671664    B1       G06F-009/455
US 20040153805 A1       H02H-003/05   Cont of application US 2000510371
                                     Cont of patent US 6671664

Abstract (Basic): DE 10058371 A1
        NOVELTY - The development of central processing units(CPUs)
    requires the application of random code generators (RCGs) for
    generating random software programs which provide the thousands of
    random commands required during testing of the CPUs. The initial
    contents of the data locations are represented by values, and the
    random program is executed by processing of commands. The various
    states of the values are then marked, and a non-fixed input value is
    propagated to a non-fixed output value, followed by conversion into a
    fixed value with a desired content.
        USE - Design verification of central processing units (CPUs) which
    carry out programs according to instruction set architecture (ISA).
        ADVANTAGE - Method of generating random programs and a method
    managing random program generation with improved test coverage
        DESCRIPTION OF DRAWING(S) - A detailed block diagram of the test
    environment of the random code generator is given.
        Random code generator (RCG) (20)
        Program counter (22)
        Processor (100)
        Integer-value register (110)
        Moving point register (120)
        Memory part-system (130)
        Accumulators (140)
        Memory control (150)
        pp; 23 DwgNo 2/12
Title Terms: MANAGEMENT; NON; FIX; REGISTER; VALUE; RANDOM; PROGRAM;
  GENERATE; DESIGN; VERIFICATION; CPU; REQUIRE; MARK; VALUE; STATE; NON;
  FIX; CONTENT; UNKNOWN; CONTENT; CAN; GENERATE
Derwent Class: S01; T01
International Patent Class (Main): G06F-009/455; G06F-011/263;  H02H-003/05

International Patent Class (Additional): G01R-031/3183
File Segment: EPI
Manual Codes (EPI/S-X): S01-G01; S01-G01A5; T01-G02A2D; T01-G07A
```

```
Set     Items   Description
S1         17   AU=(MANGELSDORF S? OR MANGELSDORF, S?)
S2          1   S1 AND (RCG OR RANDOM()(CODE? OR PROGRAM?) OR ISA)
S3          5   S1 AND (TEST OR VERIF? OR TESTING OR EMULAT? OR DEBUG? )
S4          5   S2 OR S3
S5          2   RD (unique items)
File   2:INSPEC 1969-2004/Aug W4
         (c) 2004 Institution of Electrical Engineers
File   6:NTIS 1964-2004/Aug W4
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File   8:Ei Compendex(R) 1970-2004/Aug W4
         (c) 2004 Elsevier Eng.  Info. Inc.
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Aug W5
         (c) 2004 Inst for Sci Info
File   9:Business & Industry(R) Jul/1994-2004/Sep 02
         (c) 2004  The Gale Group
File  35:Dissertation Abs Online 1861-2004/Jul
         (c) 2004 ProQuest Info&Learning
File  65:Inside Conferences 1993-2004/Aug W5
         (c) 2004 BLDSC all rts. reserv.
File 148:Gale Group Trade & Industry DB 1976-2004/Sep 03
         (c)2004 The Gale Group
File  94:JICST-EPlus 1985-2004/Aug W1
         (c)2004 Japan Science and Tech Corp(JST)
File 275:Gale Group Computer DB(TM) 1983-2004/Sep 03
         (c) 2004 The Gale Group
File 636:Gale Group Newsletter DB(TM) 1987-2004/Sep 03
         (c) 2004 The Gale Group
File 674:Computer News Fulltext 1989-2004/Aug W3
         (c) 2004 IDG Communications
File 647:CMP  Computer Fulltext 1988-2004/Aug W4
         (c) 2004 CMP Media, LLC
```

**5/5/1 (Item 1 from file: 2)**
DIALOG(R)File 2:INSPEC
(c) 2004 Institution of Electrical Engineers. All rts. reserv.

5693395 INSPEC Abstract Number: B9710-1265F-054, C9710-5130-025
**Title: Functional** verification **of the HP PA 8000 processor**
Author(s): **Mangelsdorf, S.T.** ; Gratias, R.P.; Blumberg, R.M.; Bhatia, R.
Journal: Hewlett-Packard Journal vol.48, no.4 p.22-31
Publisher: Hewlett-Packard,
Publication Date: Aug. 1997 Country of Publication: USA
CODEN: HPJOAX ISSN: 0018-1153
SICI: 0018-1153(199708)48:4L.22:FV8P;1-#
Material Identity Number: H009-97004
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A); Practical (P)

Abstract: The advanced microarchitecture of the HP PA 8000 CPU has many features that presented significant new **verification** challenges. These include out-of-order instruction execution, register renaming, speculative execution, four-way superscalar operation, decoupled instruction fetch, concurrent system bus interface, and PA-RISC 2.0 architecture enhancements. Enhanced functional **verification** tools and processes were required to address this microarchitectural complexity. (1 Refs)

Subfile: B C

Descriptors: formal **verification** ; Hewlett Packard computers; microprocessor chips; reduced instruction set computing

Identifiers: functional **verification** ; HP PA 8000 processor; microarchitecture; **verification** challenges; out-of-order instruction execution; register renaming; speculative execution; four-way superscalar operation; decoupled instruction fetch; concurrent system bus interface; PA-RISC 2.0 architecture enhancements; microarchitectural complexity

Class Codes: B1265F (Microprocessors and microcomputers); C5130 ( Microprocessor chips); C6110F (Formal methods); C5220 (Computer architecture)

Copyright 1997, IEE

**5/5/2 (Item 1 from file: 275)**
DIALOG(R)File 275:Gale Group Computer DB(TM)
(c) 2004 The Gale Group. All rts. reserv.

01531023 SUPPLIER NUMBER: 12475654 (USE FORMAT 7 OR 9 FOR FULL TEXT)
**VLSI circuits for low-end and midrange PA-RISC computers.**
**(very-large-scale-integrated chips used in HP PA-RISC reduced instruction set computing workstations)(includes related article on performance modeling and simulation) (Technical)**
Gleason, Craig A.; Johnson, Leith; **Mangelsdorf, Steven T.** ; Meyer, Thomas O.; Forsyth, Mark A
Hewlett-Packard Journal, v43, n4, p12(11)
August, 1992
DOCUMENT TYPE: Technical ISSN: 0018-1153 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 8027 LINE COUNT: 00647

ABSTRACT: A technical description of the very-large-scale-integration (VLSI) chip designs used in HP's new 9000 Series 700 workstations is presented. The chips include a 577,000-transistor CPU, a 640,000-transistor floating-point coprocessor and a 185,000-transistor memory and I/O controller and are collectively known as the PCX-S chip set. Much of the PCX-S design is based on the earlier PCX chipset used in HP's high-end multiprocessing systems; modifications were made to lower the cost. The CPU is divided into seven major blocks including two data paths, two memory-management units, a cache control programmable logic array (PLA), a pipeline control PLA and an I/O block containing interfaces to other chips. Most of its registers use transparent latches. There are full data, tag and address interfaces to separate instruction and data caches. A virtual address is used to index the static random access memory cache chips. The floating-point coprocessor (FPC) includes two data paths, a large register file, a gate array for control logic and 10 instruction pipeline registers;

it is tightly coupled to the CPU and caches. Memory and I/O chips for the PCX-S chip set are described in detail, along with specific extensions to the PA-RISC architecture used in the new systems.

SPECIAL FEATURES: illustration; chart; table; photograph
COMPANY NAMES: Hewlett-Packard Co.--Products
DESCRIPTORS: Technology; Very-Large-Scale Integration; RISC; Circuit Design; Microprocessor
SIC CODES: 3571 Electronic computers; 3577 Computer peripheral equipment, not elsewhere classified; 3674 Semiconductors and related devices
TICKER SYMBOLS: HWP
TRADE NAMES: HP Apollo 9000 700 (HP PA-RISC-based system)--Design and construction
FILE SEGMENT: CD File 275

**01541313/9**
DIALOG(R)File 275:Gale Group Computer DB(TM)

01541313 SUPPLIER NUMBER: 12653766 (THIS IS THE FULL TEXT)

**RPG development. (Harland Software's RPG Development System) (Brief Article) (Product Announcement)**

MIDRANGE Systems, v5, n19, p53(1)
Oct 13, 1992
DOCUMENT TYPE: Product Announcement ISSN: 1041-8237 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT
WORD COUNT: 202 LINE COUNT: 00016

TEXT:

Harland Software announces the release of the Harland RPG Development System for the PC, PS/2 and compatibles. Support for RPG II, RPG III, RPG/400 and SAA RPG is provided in a single product.

The Harland RPG Development System also provides file support for disk files (fixed length records, variable length records and indexed fixed length records), workstation files (using System/36 and AS/400 compatible SFGR statement), CRT device, printer files and special files. All RPG II and RPG III operation codes, including string operations, are implemented except commitment control and multiple device support. The CALL/PARM feature can call to any depth up to available memory. Other DOS programs can be executed and external subroutines written in other languages can be linked with the RPG program.

Capability for source-level debugging is provided with the Harland RPG compiler. Standard PC debuggers, such as Microsoft Codeview or Symdeb and Borland Turbo Debugger, can be used to debug programs at the source and/or symbolic level. The Harland RPG Development System is priced at $695 per machine.

Harland is located at 71-30 Kissena Blvd., Flushing, N.Y. 11367; (718) 575-5069.

Circle 436 on reader card

COMPANY NAMES: Harland Software--Product introduction
DESCRIPTORS: Product Introduction; Debugging; Application Development Software
SIC CODES: 7372 Prepackaged software
TRADE NAMES: RPG Development System (Program development software)--Product introduction
FILE SEGMENT: CD File 275

```
Set     Items   Description
S1       7473   TEST OR TESTING OR TESTED OR VERIF? OR DEBUG? OR -
             EMULAT?
S2         50   RCG OR RPG OR (PSEUDORANDOM OR RANDOM)()(PROGRAM? OR CODE?)
S3       4147   COMMIT? OR UNCOMMIT? OR DEFER? OR VALUE?
S4         30   (LENGTH? OR SIZE? OR BYTE?)(N)(RANDOM()PROGRAM? OR RANDOM(-
             )CODE? OR CODE? OR PROGRAM?) OR CODELENGTH?
S5          5   S1 (9N) S2
S6          0   S5 (S) (S3 OR S4)
S7       2938   CPU OR CPUS OR VLSI OR ASIC OR CHIP OR CHIPS OR MICROCHIP -
             OR PRESILICON OR POSTSILICON OR PROCESSOR?
S8          0   S1 AND S2 AND (S3 OR S4)
S9          1   S2 AND S7
File 256:TecInfoSource 82-2004/Jul
         (c)2004 Info.Sources Inc
```

**02104513/9**
DIALOG(R)File 275:Gale Group Computer DB(TM)

02104513 SUPPLIER NUMBER: 19758437 (THIS IS THE FULL TEXT)

**Functional verification of the HP PA 8000 processor. (Product Information)(Technical)**

Mangelsdorf, Steven T.; Gratias, Raymond P.; Blumberg, Richard M.; Bhatia, Rohit

Hewlett-Packard Journal, v48, n4, p22(10)
August, 1997
DOCUMENT TYPE: Technical ISSN: 0018-1153 LANGUAGE: English
RECORD TYPE: Fulltext; Abstract
WORD COUNT: 8154 LINE COUNT: 00655

ABSTRACT: Th almost all-new microarchitecture in the HP PA 8000 presented notable verification challenges. In the presilicon verification phase, three basic tactics were used: RTL (Register Transfer Language) simulation, accelerated simulation and switch-level simulation. The RTL verification environment comprised the PA 8000 RTL model, bus emulators, checking software and various test case sources and tools. The HP engineers chose a cycle-based accelerated simulation rather than in-circuit emulation. Accelerated simulation offers the faster speeds needed with a complex design like that of the PA 8000. Switch-level simulation checked the equivalence of the PA 8000's handcrafted transistor-level schematics with the RTL. The postsilicon verification effort centered around aggressive characterization of hardware prototypes.

TEXT:

Computer system performance has been improving recently at a rate of 40 to 60 percent per year. This growth rate has been fueled by several factors. Advancements in integrated circuit technology have made higher microprocessor clock rates and larger caches possible. There have been contributions from system software as well, such as compilers that emit more efficient machine code to realize a given function. The PA-RISC instruction set architecture has evolved to keep pace with changes in technology and customer workloads.

These factors alone, however, would not have been sufficient to satisfy customer demand for increased performance in a very competitive industry. The balance has been made up by innovations in microarchitecture that increase the amount of useful work that a microprocessor performs m a clock cycle. This has increased the complexity of the design and thus the effort required for successful functional verification.

Many of our previous microprocessor projects have reused existing cores (although generally with significant modifications and enhancements). In contrast, the HP PA 8000 CPU has a new microarchitecture that borrows little from previous projects. Some of the features in its microarchitecture presented significant new verification challenges:

* Out-of-order execution. A 56-entry queue of pending instructions is maintained by an instruction reorder buffer (IRB). The queue hardware selects instructions for execution that have their operands available irrespective of program order.

* Register Renaming. Write-after-write and write-after-read ordering dependencies are eliminated by remapping references from an architectured register to a temporary register.

* Speculative Execution. The PA 8000 predicts whether a branch is taken and can tentatively execute instructions down the predicted path. The side effects of all such instructions must be canceled if the prediction turns out to be incorrect.

* Four-way Superscalar Operation. The PA 8000 has ten functional units and can sustain an execution rate of four instructions per cycle.

* Decoupled Instruction Fetch. Instructions are fetched and inserted into the queue by an autonomous instruction fetch unit (IFU). The IFU performs branch prediction and caches the target addresses of recently taken branches in a branch target address cache (BTAC).

* Concurrent System Bus Interface. Memory requests can be issued out of order, and data returns can be accommodated out of order. Up to 16 requests can be outstanding at a time.

* PA-RISC 2.0 Architecture Enhancements. These provided important new capabilities, such as 64-bit addressing and computation, but they necessitated tool rework and limited reuse of existing test cases.

This paper describes the enhanced functional verification tools and processes that were required to address the daunting microarchitectural complexity of the PA 8000.

## Verification Overview

The purpose of functional verification is to identify defects in the design of a microprocessor that cause its behavior to deviate from what is permitted by the specification. The specification is the PA-RISC instruction set architecture and the bus protocols established by industry standards or negotiated with the designers of other system components. Performance specifications, such as instruction scheduling guidelines committed to the compiler developers, may also be considered.

Although it is not possible to prove the correctness of a microprocessor design absolutely through exhaustive simulation or existing formal verification techniques, a functional verification effort must achieve two things to be considered successful. first and foremost, it must provide high confidence that our products will meet the quality expectations of our customers. At the same time, it must identify defects early enough in the design cycle to avoid impacting the product's time to market.

A typical defect caught early in the design cycle might cost only one engineering day to debug and correct in the RTL (Register Transfer Language). Close to tape release, it might take five to ten days to modify transistor-level schematics and layout, modify interblock routing, and repeat timing analysis. Therefore, tape release can be delayed if the defect rate is not driven down quickly.

After tape release, lost calendar time is the primary cost of defects because the time required to fabricate a new revision of the design is at best a few weeks and at worst a few months. Defects that are so severe that they block a software partner's development, tuning, or testing efforts can put them on the critical schedule path of the product. The worst-case scenario is a masking defect that blocks further testing efforts for a certain functional area of the design, and this delays the discovery of additional defects by the time required to fabricate a new revision. One or more masking defects in series can quickly devastate the product schedule.

The PA 8000 verification effort consisted of a presilicon phase and a postsilicon phase. The purpose of the presilicon phase was to find defects concurrently with the design, when the cost of correcting them was small, and to drive up the quality level at first tape release so that the first prototypes would be useful to our software partners. This was done using three tactics: RTL simulation, accelerated simulation, and switch-level simulation. The postsilicon effort consisted of aggressive characterization of hardware prototypes to complete verification before systems were shipped to customers. Also, performance verification was done at various stages in the project.

## RTL Simulation

Most previous PA-RISC microprocessor projects have built their functional verification efforts around an internally developed RTL simulator that compiles RTL descriptions of blocks into C code which are then compiled with HP's C compiler. Block execution is scheduled dynamically using an event-driven algorithm. This simulation technology achieves modest performance (about 0.5 Hz running on a typical workstation), but it does provide capabilities for rapid prototyping such as the ability to simulate very high-level RTL and quick model builds. Therefore, our RTL simulator became the cornerstone of our verification effort early in the design.

Fig. 1 shows the verification environment used for RTL simulation. There are four basic components in the environment:

* The RTL model for the PA 8000.

* Bus emulators, which can apply interesting stimulus to the input buses of the PA 8000 including responses to its transactions. We included emulators for all components sharing the system bus including the memory system, 1/0 adapter, and third-party processors.

* Checking software, which monitors the behavior of the PA 8000 and verifies that it complies with the specifications. This also helps speed debugging by flagging behavioral violations as soon as they occur.

* A variety of test case sources and tools that can compile the test cases into an initial state for the PA 8000 model and configure the bus emulators.

## Checking Software

The most important check is a thorough comparison between instructions retiring in the PA 8000 model and instructions retiring in the PA-RISC architectural simulator. Retiring means exiting the instruction reorder buffer, or IRB (see article, page 8). A tool called the depiper captures information about each instruction retiring in the PA 8000 model, including what resources (such as destination registers) are being modified and the new values. The synchronizer compares this with similar information obtained from the PA-RISC architectural simulator which is also running the same test case. This provides very high confidence that the PA 8000 complies with the basic PA-RISC instruction set architecture. A final-state comparison of all processor and memory state information is also done at the end of each test case. The depiper also provides the synchronizer with information about architecturally transparent events such as cache misses. Using this information, the synchronizer can perform strong cheeks in the areas of cache coherency, memory access ordering consistency, and memory-to-cache transfers. In addition, a number of checkers were developed for other areas:

* A checker for the instruction queues, including whether the order in which instructions are sent to functional units complies with data dependencies
* A checker for protocol violations on the system bus
* A checker for the bus interface block, discussed in more detail below
* A checker that detects unknown (X) values on internal nodes.

## Test Case Sources

A test case is essentially a test program to be run through the RTL model of the processor to stress a particular area of functionality. These are generally written in a format similar to PA-RISC assembly language, with annotations to help specify initial cache and TLB contents. In addition, a control file can be attached to a test case to specify the behavior of the bus emulators. The emulators have useful default behavior, but if desired the control files can precisely control transaction timing.

A test case is compiled using a collection of tools that includes the PA-RISC assembler. The result of the compilation is a set of state initializations for the RTL model. These include the processor registers, caches, TLB, and memory. In addition, the bus emulators are initialized with the commands they will use during execution of the test case.

Previous PA-RISC microprocessor projects had built up a library of test cases and architectural verification program (AVPs). Although we did run these, it was clear from the beginning that a large source of new cases would be required. The existing cases were very short, so their ability to provide even accidental coverage for a machine with a 56-entry IRB was questionable. Moreover, we needed cases that targeted the unique microarchitectural features of the PA 8000.

We developed a test case template expander to improve our productivity in generating the large number of cases required. An engineer could write a test template specifying a fundamental interaction, and the tool would expand this into a family of test cases. Some of the features of this tool included:

* The ability to sweep a parameter value. This was often used to vary the distance between two interacting instructions.
* The ability to fill in an unspecified parameter with a random value.
* An if construct, so that a choice between two alternatives could be conditional on parameters already chosen.
* Instruction groups, so that an instruction could be specified that had certain characteristics without specifying the exact instruction.

We also used the pseudorandom code generator and test coverage measurement techniques discussed below in the RTL simulation environment. To improve our coverage of multiprocessor functionality, we configured our bus emulators to generate random (but interacting) bus traffic.

## Structural Verification

A block can be described by a single large RTL procedure or by a schematic that shows the interconnection of several smaller blocks, each of which is described by RTL. At the beginning of the project, RTL tends to be written at a high level because it can simulate faster and is easier to write, debug, and maintain when the design is evolving rapidly. Block designers, however, have a need to create schematics for their blocks, so there is a risk that these will diverge from the RTL reference.

We considered three strategies to verify that the two representations of the block were equivalent. The first, formal verification, was not pursued because the required tools were not yet available from external vendors. The second strategy was to rely on the switch-level verification effort. This was unattractive because defects would be found too late in the design cycle, and the planned number of vectors to be run might not have provided enough coverage. The strategy selected was to retire the higher-level RTL description and replace it in the RTL model with the lower-level representation. The more timely and thorough verification that this provided compensated for some disadvantages, including slower simulation and more difficulty in making changes. We also used this strategy selectively, relying on switch-level simulation to cover regular blocks such as data paths with little risk.

### Divide and Conquer

In any large design effort, one faces a choice of whether to verify components individually, together, or both. Verifying a component separately has several potential advantages. Simulation time is greatly reduced. Input buses can be directly controlled, so effort need not be expended manipulating the larger model to provide interesting stimulus. Finally, dependencies between subprojects are eliminated.

For separate verification to succeed, the interfaces to other components must be very well-specified and clearly documented. Investments must be made in a test jig to provide stimulus to the component and in checking software to verify its outputs. In addition, some portion of the verification must be repeated with all components integrated to guard against errors in the specifications or different interpretations of them.

The PA 8000's bus interface block was particularly well-suited to separate verification. The block had clean external interfaces but contained a lot of complexity, including the hardware to manage multiple pending memory accesses. A software checking tool was written to monitor the block's interfaces and verify its operation. Checking that a request on one bus ultimately results in a transaction on the other bus is a simple example of numerous checks performed by this tool. A very low defect rate demonstrated the success of the divide-and-conquer strategy for this block.

Most of our remaining verification effort was focused on the complete PA 8000. As a final check, a system-level RTL model was built that included several processors, the memory controller, the 1/0 adapter and other components. Although throughput was very low, basic interactions between the components were verified using this model.

### Accelerated Simulation

The speed of the RTL simulator was adequate to provide quick feedback on changes and for basic regression testing, but we lacked confidence that on a design as complex as the PA 8000 it would be sufficient to deliver an adequate quality level. We saw a strong need for a simulation capability that was several orders of magnitude faster so that we could run enough test cases to ferret out more subtle defects. We considered two technologies to provide this: cycle-based simulation and in-circuit emulation.

Cycle-based simulation provides a much faster software simulation of the design. With an event-driven simulator such as our RTL simulator, a signal transition causes all blocks that the signal drives to be reexecuted, and any transitions on the outputs of these blocks are similarly propagated until all signals are stable. The overhead to process every signal transition, or event, is fairly high. Cycle-based simulators greatly improve performance by eliminating this overhead. The design is compiled into a long sequence of Boolean operations on signal values (AND, OR, etc.), and execution of this sequence simulates the operation of the logic in a clock cycle. The name cycle-based simulator comes from the fact that the signal state is only computed at the ends of clock cycles, with no attempt to simulate intermediate timing information. Our investigation revealed that speedups of 500 times were possible, so a simulation farm of 100 machines could have a throughput on the order of 25,000 Hz. The biggest drawback of this strategy was that cycle-based simulators were not yet available from external vendors.

With in-circuit emulation, the gates in a Boolean representation of the design are mapped onto a reconfigurable array of field-programmable gate arrays (FPGAS). The design is essentially built using FPGAS, and the emulated processor is connected to the processor socket in an actual system. The clock rate of the emulation system is on the order of 300,000 Hz, so very high test throughput is possible. It is even possible to boot the operating system. Unfortunately, there were many issues involved in using in-circuit emulation successfully:

* Custom printed circuit boards would have to be designed for the caches, large register files, and any other regular structures that consume too much emulation capacity. Changes in the design would be difficult to accommodate.

* A system was needed to exercise the emulated processor, including a memory controller and 110 devices. Firmware and hardware tinkering would have been needed to make this system functional at the slow clock rates required by the emulation system.

* Productivity was reduced by long compile times and limited observability of internal signals. Only one engineer at a time could use the system for debugging.

* The strategy was difficult to extend to multiprocessor testing. It was prohibitively expensive to emulate multiple processors. We planned to use a software emulator to create third-party bus traffic and verify the processor's responses, but there was a risk that the software's performance would throttle the emulation system's clock rate. The emulation system was a very large capital investment.

We were quite wary of in-circuit emulation since its use on a previous project had failed to make a significant contribution to functional verification. We were also willing to give up the performance advantage of in-circuit emulation to avoid tackling the ease-of-use issues. The decision to use cycle-based simulation would have been simple except that it meant that we would have to develop the simulator ourselves. R&D organizations in HP are challenged to focus on areas of core competency and look to external vendors to fulfill needs such as design tools that are common in the industry. We did select cycle-based simulation because we were confident that its lower risk and higher productivity would translate into a competitive advantage.

We were careful to reuse components wherever possible and to limit the scope of the project to providing the tool functionality required to verify the PA 8000. We did not attempt to create a simulation product useful to other groups within HP. This turned out to be a good decision because comparable tools have recently started to become available from external vendors.

Cycle-Based Simulation Compiler

The cycle-based simulation compiler operates only on simple gate-level primitives such as logic gates and latches, so higher-level RTL must first be synthesized into a gate-level equivalent. We had to develop our own translator for this because the RTL language used by our RTL simulator was defined before the industry standardization of such languages. Another simplification is that signal values are limited to 0 and 1, with no attempt to model an unknown (X) state.

Fig. 2 shows a simple example circuit, a two-bit counter, that we will use to illustrate the compilation process. The user must describe to the compiler information about the circuit's clocks. The clock cycle is broken down into two or more phases, with the state of the clocks fixed during each phase. This circuit has a clock cycle of two phases, and the clock (CLK) is low during the first phase and high during the second phase.

The compiler uses this information to determine which gates need to be evaluated during each phase. This is done in two steps. First, for each phase, the compiler propagates the clock values into the circuit. This uses simple rules of Boolean logic, such as the fact that the output of an AND gate with a zero input must be zero. The goal is to identify latches with a zero control, which are therefore provably opaque during that phase. Next, again for each phase, the compiler finds all gates that can be reached from a clock or other input through a path that does not contain an opaque latch.

Next, a sequence of Boolean operations is emitted corresponding to the gates in each phase. Because we used PA-RISC machines for simulation, the sequences were actually output in PA-RISC assembly language. The sequences totaled more than two million instructions for the PA 8000 design. The gates are ordered in sequence so that a gate is not emitted until its inputs have been computed. Cycles, or loops, m the circuit are handled by looping through the gates in the cycle until all circuit nodes are stable.

Numerous optimizations are done on the output assembly language sequences:

* Clock signals have known values during each phase, which can be propagated into the circuit. These constant values can simplify or eliminate some of the Boolean operations.

* The 32 PA-RISC registers are used to minimize loads and stores to memory. Boolean operation scheduling and victim register selection are employed to minimize the number of loads and stores.

* The compiler can determine which circuit nodes carry information from one phase to the next. The remaining nodes are temporaries whose values need not be flushed to memory after their final use within a phase.

* To eliminate NOT operations corresponding to inverting gates, the compiler can represent nodes in inverted form and perform Demorgan transformations of Boolean operations (e.g., NOT-AND is equivalent to OR-NOT).

* Aliasing of circuit nodes is done to eliminate code for simple buffers and inverters.

Any one of the Boolean operations in the output assembly language sequence operates on all 32 bits of the PA-RISC data path, as shown in Fig. 3. We make use of the parallelism to run 32 independent test cases in parallel. This is possible because the simulator always executes exactly the same sequence of assembly language instructions regardless of the test case (assuming the circuit being simulated is the same). This does not reduce the time to solution for a given test case, but it does increase the effective throughput of the simulator by 32 times. This was still very useful because our verification test suites are divided into a vast number of fairly short test cases.

The compiler allows the user to write C++ behavioral descriptions of blocks such as memories and register files that are not efficient to represent using gate-level primitives. The compiler automatically schedules the calls to this C++ code, and an API (application programming interface) gives the code access to the block's ports.

## Pseudorandom Testing

We had learned from previous projects that the type of defects likely to escape the RTL simulation effort would involve subtle interactions among pending instructions and external bus events. With up to 56 instructions pending inside the processor and a highly concurrent system bus with multiprocessing support, k is not possible to count -- much less fully test -- all of the interactions that might occur We believed that the value of handwritten test cases and test cases randomly expanded from templates was reaching diminishing returns, even with the low simulation throughput achievable with the RTL simulator.

We had also learned that pseudorandom code generators were a very effective means of finding these kinds of defects. Such a program generates a pseudorandom sequence of instructions that use pseudorandom memory addresses and pseudorandom data patterns. However, it is important that the program make pseudorandom selections in a manner that considers the microarchitecture of the processor and the kinds of interaction defects that are likely to occur.

Selecting memory addresses is a good example. Memory addresses are 64 bits wide. If they were selected truly randomly, reusing the same address within a test case would be an impossibly rare event. This would fail to stress important aspects of the machine, such as the logic that detects that a load is dependent on a preceding store with the same address. There are hundreds of selections that a generator makes in which the microarchitecture must be carefully considered.

We chose to target the cycle-based simulation environment for a new pseudorandom code generator. Our pseudorandom code generator was carefully tuned for the microarchitecture of the PA 8000 and included support for the new PA-RISC 2.0 instruction set. Hundreds of event probabilities could be specified by a control file to provide engineering control over the types of cases being generated. We also chose not to port the rich set of checking software from the RTL simulation environment to the cycle-based simulation environment because of the effort involved and risk that performance would be reduced. Generators such as our pseudorandom code generator predict the final register and memory state of the processor, and defects will generally manifest themselves as mismatches between the simulated and predicted final state. It is possible that an error in state will be overwritten before the end of a test case, but a defect won't be missed unless this happens in every test case that hits it, which is extremely unlikely statistically. Our experience with hardware prototype testing, in which internal signals are unavailable and all checking must be done through final state, also made us confident in this strategy.

### Cycle-Based Simulation Environment

Fig. 4 shows the cycle-based simulation environment, which will be described by following the life cycle of a typical test case. The job controller controls the 32 independent simulations that are running in the data path positions, or slots, of the cycle-based simulation model. It starts and ends test cases in the 32 slots independently. It is controlled by a UNIX(R) shell, which is driven either by a script or interactively for debug activities.

When a slot becomes available, the controller commands the pseudorandom code generator to generate a new test case, occasionally first reading a new control file. The test case is specified by the initial state of memory and the processor's registers, and the pseudorandom code generator specifies the initial state of the caches as well to prevent an initial flurry of misses.

The pseudorandom code generator downloads the initial state of the simulation into various components of the simulated model. These include the gate-level model, behavioral models representing caches, register files, and other regular structures, and emulators representing bus devices such as the memory system, I/O adapter, and third-party processors. The model is then stepped for numerous clock cycles until a breakpoint trigger fires to indicate the end of the test case. The pseudorandom code generator is then commanded to extract the relevant final state from the simulated model and compare it with the final state that it predicted to determine whether the test case passed.

We used a simulation farm of up to 100 desktop workstations and servers for cycle-based simulation. Jobs were dispatched to these machines under the control of HP Task Broker.(1) Each job ran several thousand test cases that were generated using a specific pseudorandom code generator control file.

### Multiprocessor Testing

Multiprocessor testing was a key focus area. We wrote emulators for additional processors and the I/O adapter, which share the memory bus. It was only necessary to emulate the functionality required to initiate and respond to bus transactions, but the emulators were accurate enough that defects in the processor related to cache coherency would manifest themselves as mismatches in the final memory state.

We established linkages with our pseudorandom code generator so that the emulators would be more effective. When a test case started, the pseudorandom code generator downloaded control file information so that parameters such as transaction density and reply times could be easily varied. The pseudorandom code generator also downloaded the memory addresses used by the test case so that the emulators could initiate transactions that were likely to cause interactions.

### Coverage Improvement

Improving the test coverage of our pseudorandom code generator was an ongoing activity. The pseudorandom code generator has hundreds of adjustable values, or knobs, in its control file, which can be varied to focus the generated test cases. We found that the defect rate quickly fell off when all knobs were left at their default settings.

We used two tactics to create more effective control files. First, we handcrafted files to stress particular functional areas. Second, we generated files using pseudorandom techniques from templates, each template specifying a particular random distribution for each knob. We found with both strategies that it was important to monitor the quality of the files generated.

We did this in two ways. First, our pseudorandom code generator itself reported statistics on the test cases generated with a given control file. A good example is the frequency of traps. Traps cause a large-scale reset inside the processor, including flushing the instruction queues, so having too many traps in a case effectively shortens it and reduces its value. We made use of instrumentation like this to steer the generation of control files.

Feedback is often needed based on events occurring within the processor, which our pseudorandom code generator cannot generally predict. For example, an engineer might need to know how often the maximum number of cache misses are pending to be confident that a certain area of logic has been well-tested. Test case coverage analysis was accomplished by an add-on tool in the simulation environment. This tool included a basic language that allowed engineers to describe events of interest using Boolean equations and timing delays. The list of events could include those that were expected to occur regularly or even those that a designer never expected to occur. Both ends of this spectrum could provide useful information.

Once the events were defined, the add-on tool provided monitoring capabilities during the simulation. As test cases were run, the tool would generate output every time it detected a defined event. This output was then postprocessed and assembled into an event database. The event database could contain results of thousands of test case runs. Event activity reports were then generated from this event database. These reports included statistics such as frequency of events, duration of events, the average, maximum, and minimum distance between two occurrences of a event, and so on.

The event activity reports were then analyzed by engineers to identify weak spots in coverage and provide feedback to the generation of control files. This methodology provided one other benefit as well. For many functional defects, especially ones that were hard to hit, the conditions required to manifest the defect were coded and defined as an event. Then this add-on tool was used with a model that contained a fix for the defect to prove that the conditions required for the defect were being generated.

### Switch-Level Simulation

In typical ASIC design methodologies, an RTL description is the source code for the design, and tools are used to synthesize transistor-level schematics and IC layout mechanically from the RTL. Verifying the equivalence of the synthesized design and the RTL is largely a formality to guard against occasional tool defects. In the full-custom methodology used on the PA 8000, however, designers handcraft transistor-level schematics to optimize clock rate, die area, and power dissipation. Therefore, we needed a methodology to prove equivalence of the handcrafted schematics and the RTL.

At the time the project was undertaken, formal verification tools to prove this equivalence were not available. Instead, we turned to an internally developed switch-level simulator. Although much slower than the RTL simulator, the switch-level simulator included essential features such as the ability to model bidirectional transistors, variable drive strength, and variable charge ratios. Thanks to this careful effort in switch-level verification on the PA 8000, not a single defect was found on silicon that was related to a difference between the transistor-level schematics and the RTL.

Verification was performed by proving that a block behaved the same when running a test case in the RTL simulator and in the switch-level simulator. First, a full-chip RTL simulation of a test case was done with the ports of a block monitored. These vectors were then turned into stimulus and assertions for a switch-level simulation of the block. Initializing the state of the block identically in the two environments was a challenge, especially since the hierarchies and signal names of the RTL and schematic representations can differ.

Initially, this strategy was used to turn on the switch-level simulator models of individual blocks on the chip. This helped to distribute the debug effort and quickly bring all blocks up to a reasonable quality level. Afterward, the focus shifted to full-chip switch-level simulator verification. In addition to collecting vectors at the ports of the chip, thousands of internal signals were monitored in the RTL simulation and transformed into assertions for the switch-level simulation. These were valuable for debugging and raising our confidence that there were no subtle behavioral differences between the two models.

The RTL simulation effort was a plentiful source of test cases, but they were targeted at functional defects rather than implementation errors, and the slower speed of the switch-level simulator allowed only a portion of them to be run. To improve coverage, the process shown in Fig. 5 was used at the block level. The RTL description for the block was converted into an equivalent gate-level model using tools developed for cycle-based simulation. Automated test generation tools, normally used later in the project for manufacturing, were then used to create test vectors for the gate-level model. If the switch-level simulation using these vectors failed, then the two representations were known to differ. While the automated test generation tools do not generate perfect test vectors, this process still proved to be a valuable source of additional coverage.

The switch-level simulator also supports several different kinds of quality checks. These include dynamic decay checking to detect undriven nodes, drive flght checking to detect when multiple gates are driving the same node, and a race checking methodology. This was implemented by altering how the clock generator circuits were modeled to create overlap between the different clocks on the chip. Failures that arose from overlapped clocks pointed to paths requiring detailed SPICE simulations to prove that the race could not occur in the real circuits. Reset simulations were done from random initial states to ensure that the chip would power up properly. Finally, a switch-level simulator model was built from artwork netlists to prove that there were no mismatches between the artwork and the schematics that were missed by other tools.

## Postsilicon Verification

Presilicon verification techniques are adequate to find nearly all of the defects in a design and bring the level of quality up to the point where the first prototypes are useful to software partners. However, defects can be found in postsilicon verification that eluded presilicon verification for many reasons.

First, test code can be run at hardware speeds, several orders of magnitude faster than even the fastest simulators. Errors in the simulation models or limitations in the simulators themselves can cause the behavior of the silicon to differ from that predicted by the simulators. Finally, most simulation is targeted at system components, such as the PA 8000 itself, rather than the entire system. Errors in the specifications for interfaces between components, or different interpretations of these specifications, can become apparent when all components are integrated and the system exercised.

## Overlapped Test Coverage

Previous projects had established the value of running test code from as many sources as possible. Each test effort had its own focus and unique value, but each also had its own blind spots. This is even true for pseudorandom code generators. In the design of these very complex programs, many decisions are made that affect the character and style of the generated code. There can be code defects as well that cause coverage holes. The large overlap in coverage between efforts proved to be an invaluable safety net against the limitations and blind spots of individual tools.

The PA 8000 verification team focused its effort on pseudorandom code testing. Experience showed that this would be the primary source of subtle defects and would allow us to find most defects before our software partners. We ran several tools including our pseudorandom code generator and generators used in the development of the PA 7200 and PA 7300LC processors and the HP 9000 Model 725 workstation. Several tools were capable of generating true multiprocessing test cases that included data sharing between random sequences running on different processors. Data sharing with DMA processes was implemented as well.

We developed a common test environment for most of these random code generators. The HP-UX(*) operating system is not a suitable environment because its protection checks do not permit many of the processor resources to be easily manipulated. Our test environment allowed random testing of privileged operations and also included many features to improve repeatability and facilitate debugging. For example, it performed careful initialization before each test case so that, aided by logic analyzer traces, we could move a failing test case to the RTL simulator for easy debugging (in hardware, there is no access to internal signals). We established a plug-and-play API so that the investment in the environment could be leveraged across several generators.

In parallel with our pseudorandom testing, our software partners pursued their own testing efforts. While primarily targeted at their own software, this provided stress for the processor as well. The test efforts included the HP-UX and MPE/XL operating system kernels, I/O and network drivers, commands, libraries, and compilers. Performance testing also provided coverage of benchmarks and key applications. Finally, although it did not find any defects in the PA 8000, HP's Early Access Program made available preproduction units to customers and external application developers.

### Ongoing Improvement

When defects were found, we used the process shown in Fig. 6 to learn as much as possible about why the defect was missed previously and how coverage could be improved to find additional related defects. After the root cause of the defect was determined, actions were taken in the areas of design, presilicon verification, and postsilicon verification.

The designers would identify workarounds for the defect and communicate these to our software partners, at the same time seeking their input to evaluate the urgency for a tape release to fix the defect. The design fix was also determined, and inspections were done to validate the fix and brainstorm for similar defects.

In the area of presilicon verification, reasons why the defect was missed would be assessed. This usually turned out to be a test case coverage problem or a blind spot in the checking software. Models would then be built with the design fix and other corrections. Test coverage would be enhanced in the area of the defect, and simulations were done to prove the fix and search for any related defects. Cycle-based simulation played a big role here by finding several introduced defects and incomplete fixes.

The postsilicon verification activities were similar. Coverage for the tool that found the defect would be enhanced, either by focusing it with control files or by tool improvements. Spurred by a healthy rivalry, engineers who owned other tools would frequently improve them to show that they could hit the defect as well. All of this contributed to finding related defects.

### Performance Verification

At the time the PA 8000 was introduced in products, it was the world's fastest available microprocessor. Careful microarchitectural optimization and verification of the design against performance specifications were factors in achieving this leadership performance.

In a microarchitectural design as complex as the PA 8000, seemingly obscure definition decisions and deviations of the design from the specification can cause a significant loss of performance when system-level effects and a variety of workloads are considered. A good example is a design defect that was found and corrected in the PA 8000. When a cache miss occurred under certain circumstances, a dirty cache line being evicted from the cache would be written out on the system bus before the read request for the missing line was issued. Since the addresses of the two lines have similar low-order bits, both mapped to the same bank of main memory. The memory controller would begin processing the write as soon as it was visible on the bus, busying the memory bank and delaying the processing of the more critical read.

A detailed microarchitectural performance simulator was written early in the project to help guard against such issues. It was used to project performance and generate a statistical profile for a variety of benchmarks and applications. Workloads with surprising results or anomalous statistics were targeted for more detailed analysis, and through this process opportunities were identified to improve the microarchitecture. Particularly valuable feedback came from the compiler development team, who used the simulator to evaluate the performance of compiler prototypes. The concurrent development of tuned compilers with close cooperation between hardware and software teams was a key contributor to the PA 8000's performance leadership.

The microarchitectural performance simulator was written at a somewhat abstract level, so it could not provide feedback on whether the detailed design met the performance specifications. Comparing the performance of the RTL simulator against the microarchitectural simulator was the obvious way to address this, but the RTL simulator was far too slow. As a compromise, we performed this comparison on key performance kernels that were tractable enough for the RTL simulator. We also developed a path by which a workload could be run up to a critical point using the microarchitectural simulator, at which point the state of the memory, caches, and processor registers could be transferred into the RTL simulator for detailed simulation.

Performance verification continued in the postsilicon phase of the project. The PA 8000 incorporated several performance counters that could be configured to count numerous events. These were used to help identify workloads or segments of workloads needing closer analysis. The PA 8000's external pins and debug port provided sufficient information to determine when instructions were fetched, issued for execution, and retired. Isolation of specific performance issues was aided by a software tool called the depiper which presented a visual picture of instruction execution. Through these efforts, several performance-related hardware defects were identified and corrected before production.

## Results

Achieving a defect-free design at first tape release is not a realistic expectation for a design as complex as the PA 8000. Nevertheless, we were extremely satisfied with the quality we achieved at first tape release. The first prototypes were capable of booting the operating system and running virtually any application correctly. In fact, only one defect was ever hit by an application, although a few defects were encountered in stress testing of system software.

Fig. 7 shows the sources of defects found and corrected after first tape release. Surprisingly, about a third of the total defects were found by continued use of the presilicon verification tools (mostly the cycle-based simulation environment) for a few months following tape release. This indicates that despite the outstanding performance of cycle-based simulation, the project would have benefited from even more throughput, or perhaps use of the tool earlier. A third of the defects were also found by one of the pseudorandom code generators running on hardware prototypes. Inspections were a significant source of defects. The remaining defects were split between turn-on work, performance analysis work, and partner software testing. Since very few defects were discovered by partners, we could generally communicate workarounds ahead of time and take other steps to minimize the impact.

Fig. 8 shows the impact of the defects found after first tape release on our software partners. A large majority were never seen outside the environment in which they were found and had no significant impact. About half of these involved functional areas, such as debugging hardware, that are not even visible to applications or system software. Most of the remaining defects had only a moderate impact. Examples are defects that were found by a partner at the expense of their testing resources, defects that required a workaround in system software, and defects that required certain performance-related features in the processor to be disabled. Only a handful of defects were severe enough to temporarily block or significantly disrupt a partner's development and testing efforts. All but one of these were early multiprocessing defects that slightly delayed bringing up the multiprocessing operating system.

## Cycle-Based Simulation Results

The cycle-based simulation effort made an essential contribution to the verification of the PA 8000. Fig. 9 shows the sources of defects that eluded our RTL simulation effort (which incorporated existing best practices). If we had not made the investment in cycle-based simulation, the number of defects that would have had to be found by techniques would have been three times higher. It was much less expensive to fix the defects caught by cycle-based simulation as the design progressed than it would have been to fix them in later revisions.

Also, because cycle-based simulation tended to find the most severe defects early, no masking defects were present, and the number of serious blocking defects that we had to manage after the first tape release was reduced by three to six times. If our software partners had been exposed to this level of severe defects, it is probable that the product's time to market would have been impacted.

Finally, cycle-based simulation provided a high-confidence regression test before each tape release. Several incomplete bug fixes and new defects that had been introduced in the design were found in time to be corrected before a tape release.

## Conclusions

Continuous innovation in functional verification tools and processes is required to keep pace with the increasing micro-architectural complexity of today's CPUs. This paper has described the methodologies used to verify the PA 8000. These met our most important goal of improving the quality of the PA 8000 to the high level demanded by our customers. By finding defects early, they also helped us conserve our engineering resources and quickly deliver the industry leading performance the PA 8000.

## Acknowledgments

The authors would like to thank all of the different teams who contributed to the successful functional verification of the PA 8000. Special thanks to all of the individuals in the Engineering Systems Laboratory in Ft. Collins, Colorado and the General Systems Laboratory in Roseviue, California who were involved in the design and verification effort. Many thanks also to our partners: the Cupertino Open Systems Laboratory for testing efforts and flexibility in working around defects, the Systems Performance Laboratory and Cupertino Language Laboratory for performance verification work, and the Integrated Circuit Business Division for supporting our simulation tools.

## Reference

(1.) TR Graf, et al, "HP Task Broker: A Tool for Distributing Computational Tasks," Hewlett-Packard Journal, Vol. 44, no. 4, August 1993. pp. 15-22.

SPECIAL FEATURES: chart; illustration
DESCRIPTORS: Product Description/Specification; Microprocessor
PRODUCT/INDUSTRY NAMES: 3674124 (Microprocessor Chips)
SIC CODES: 3674 Semiconductors and related devices
TRADE NAMES: HP PA-RISC PA-8000 (Microprocessor)--Design and construction
FILE SEGMENT: CD File 275

United States

» HP Home | » Products & Services | » Support & Drivers | » Solutions | » How to B

» Contact HP

Search:

● HP Labs ○ All of HP US

invent

# HP Labs Technical Reports Search

» HP Labs
» Research

» Advanced studies
» Internet & computing platforms
» Printing & imaging
» Solutions & services

» News and events
» **Technical reports**

» About HP Labs
» People
» Worldwide sites

» Downloads

## Grouped by ● Confidence ○ Subject

Your concept search was: **rpg commit verification**

New Search

- Scores with a red icon show confidence in the match between the document and your search.
- Search for similar documents by clicking on the red or black icons next to each score.

Documents found by matching keyword prefixes and concept-based associations:

65% **HP Labs : Tech Report: HPL-2001-317: Quantum Bit String Commitment**
63% **Tech Report: HPL-2000-33: Formal Automatic Verification of**
59% **Tech Report: CRL-93-1: A new presumed commit**
59% **Tech Report: HPL-1999-55: Low Commitment Spectrophotometer Care**
59% **HP Labs : Tech Report: HPL-2001-20: CTP: An Optimistic Commit**
57% **HP Labs : Tech Report: HPL-2001-129: Cross-Partition Protocols in a**
55% **HP Labs : Tech Report: HPL-2001-34: An Approach to Optimistic**
55% **Tech Report: CRL-90-10: Analyzing distributed commitment by**
52% **Tech Report: SRC-RR-176: Verifying Sequential Consistency on**
52% **Tech Report: SRC-RR-78: Using transformations and verification**
52% **Tech Report: HPL-2004-7: A New SVM Approach**
51% **Tech Report: CRL-90-3: Consistent timestamping for transactions**
50% **Tech Report: CRL-91-3: A transactional model for**
50% **Tech Report: SRC-RR-96: How to make a**
50% **Tech Report: WRL-95-9: Memory Consistency Models for**
50% **Tech Report: CRL-92-7: Transactional memory : architectural**
50% **Tech Report: SRC-TN-2001-003: Thread-Modular Verification For Shared-Memory**
50% **Tech Report: SRC-RR-41: Evaluating the performance of**

**50% Tech Report: SRC-RR-93: Experiences with software specification**

**50% Tech Report: SRC-RR-89: Compositional refinement of interactive**

**49% Tech Report: SRC-RR-51: Experience with the firefly**

**49% Tech Report: SRC-RR-26: Parallel compilation on a**

**47% Tech Report: HPL-91-12: A CSP Model of**

**47% Tech Report: HPL-2003-142: Signing RDF Graphs**

**44% Tech Reports: HPL-97-118: Robust Public Key Watermarking**

**44% Tech Report: HPL-2004-4: A Kullback-Leibler Divergence Based**

**44% HP Labs : Tech Report: HPL-2001-316: Founding Mistrustful Quantum Cryptography**

**44% Tech Report: HPL-92-60: A Performance Analysis of**

**44% Tech Report: HPL-2003-235: Computing the digest of**

**43% Tech Report: HPL-2003-235R1: Computing the digest of**

**43% HP Labs : Tech Report: HPL-2001-284: A Cautionary Note Regarding**

**42% Tech Report: HPL-92-58: S++ - A New**

**41% Tech Report: HPL-2000-155: Verifiable Partial Escrow of**

**40% Tech Report: HPL-94-100: Enterprise Modeling and Simulation:**

**40% Tech Report: TR-88.1: Group Commit Timers and**

**40% Tech Reports: HPL-97-123: Making an Empty Promise**

**39% Tech Report: HPL-1999-30R1: Fast Monte-Carlo Primality Evidence**

**39% Tech Report: HPL-98-203: SLA Management in Federated**

**39% Tech Report: HPL-94-39: Timing Analysis of Digital**

**39% Tech Report: HPL-IRI-98-003: Quality of Service Agents**

**38% Tech Report: HPL-94-89: Enterprise Modeling System: Inventory**

**37% Tech Reports: HPL-97-108: Security Risk Control of**

**37% Tech Report: HPL-2003-88R1: The Lifestyles of Working**

**36% HP Labs : Technical Reports : Compaq & DEC**

**36% HP Labs : Technical Reports : Compaq & DEC Technical Reports**

**36% Tech Report: HPL-90-17: Tuning the Reactivity of**

**36% HP Labs : Tech Report: HPL-2001-36: New Zero-knowledge Undeniable Signatures**

**35% Tech Report: HPL-91-170: Case Study of Object-Oriented**

**35% HP Labs : Tech Report: HPL-2002-187: Rememberer: A Tool for**

**35% HP Labs : Tech Report: HPL-2001-285: Business Process Simulation with**

**35% HP Labs : Tech Report: HPL-2002-133: An Agent-based Framework for**

**35% Tech Report: HPL-BRIMS-96-26: Insecurity of Quantum Secure**

**34% Tech Report: HPL-1999-2: Spectrophotometer Calibration**

**34% Tech Report: HPL-93-50: DS Spread Spectrum Link**

**34% HP Labs : Tech Report: HPL-2002-324: Specifying and Monitoring Guarantees**

**34% HP Labs : Tech Report: HPL-2002-294: Text Extraction via an**
**34% Tech Report: HPL-91-20: Incremental, Bottom-Up, Well-Founded Deduction**
**34% Tech Report: HPL-91-20: Incremental, Bottom-Up, Well-Founded Deduction**
**33% HP Labs : Tech Report: HPL-2001-37: Timed-Release Cryptography**
**33% HP Labs : Tech Report: HPL-2001-76: Recovery of Memory and**
**32% Tech Report: HPL-2003-173: Using Semantic Web Technology**
**32% Tech Report: HPL-97-53: Digital Color Cameras -**
**31% Tech Report: HPL-2004-95: RDF Graph Digest Techniques**
**31% Tech Report: HPL-2003-212: Applications of Coherent Population**
**31% HP Labs : Tech Report: HPL-2001-266: Quantum computation with coherent**
**30% Tech Report: HPL-1999-158: MIND--A New Psychophysical Algorithm**
**29% Tech Report: TR-88.6: A Comparison of the**
**29% Tech Report: HPL-2004-93: Direct Anonymous Attestation**
**28% HP Labs : Technical Reports : Tandem**
**28% HP Labs : Technical Reports : Tandem**
**28% HP Labs : Tech Report: HPL-2002-331: Analysis of Packett Loss**
**28% Tech Report: HPL-1999-87: Using Walsh Code Selection**
**28% Tech Report: HPL-1999-149: Design and Performance of**
**26% Tech Report: HPL-2003-254: On the Optimality of**
**26% HP Labs : Tech Report: HPL-2001-131: Composite Filter Pattern**
**25% HP Labs : Technical Reports : 2001**
**25% HP Labs : Technical Reports : 2001**
**25% HP Labs : Technical Reports : 2001**
**24% Tech Reports: HPL-98-10: QML: A Language for**
**24% Tech Report: HPL-96-71: Blind Certification of Public**
**23% Tech Report: HPL-2000-118: Two Topics in Hyperelliptic**
**23% Tech Report: HPL-2003-254: On the Optimality of**
**18% HP Labs : Technical Reports : 2000**
**18% HP Labs : Technical Reports : 2000**
**18% HP Labs : Technical Reports : 2000**
**15% HP Labs : Technical Reports : 1999**
**15% HP Labs : Technical Reports : 1999**
**12% HP Labs : Technical Reports : 2004**
**12% HP Labs : Technical Reports : 2004**
**04% DELI: A J2EE DElivery context LIbrary for CC/PP and UAProf**
**02% [Technical Report HPL-2001-190]**

Results by excite

🖶 **Printable version**

Privacy statement

Using this site means you accept its terms

Feedback to HP Labs